UNITED STATES PATENT AND TRADEMARK OFFICE

UNITED STATES DEPARTMENT OF COMMERCE
United States Patent and Trademark Office
Address: COMMISSIONER FOR PATENTS
P.O. Box 1450
Alexandria, Virginia 22313-1450
www.uspto.gov

APPLICATION NO.	FILING DATE	FIRST NAMED INVENTOR	ATTORNEY DOCKET NO.	CONFIRMATION NO.
10/696,867	10/30/2003	Conor Morrison	03797.00637	8068

28319 7590 01/04/2008

BANNER & WITCOFF, LTD.
ATTORNEYS FOR CLIENT NOS. 003797 & 013797
1100 13th STREET, N.W.
SUITE 1200
WASHINGTON, DC 20005-4051

EXAMINER	
HERBST, RACHEL M	
ART UNIT	**PAPER NUMBER**
4121	

MAIL DATE	DELIVERY MODE
01/04/2008	PAPER

Please find below and/or attached an Office communication concerning this application or proceeding.

The time period for reply, if any, is set in the attached communication.

Office Action Summary	Application No.	Applicant(s)	
	10/696,867	MORRISON ET AL.	
	Examiner	Art Unit	
	R M. HERBST	4121	

-- The MAILING DATE of this communication appears on the cover sheet with the correspondence address --

Period for Reply

A SHORTENED STATUTORY PERIOD FOR REPLY IS SET TO EXPIRE *3* MONTH(S) OR THIRTY (30) DAYS, WHICHEVER IS LONGER, FROM THE MAILING DATE OF THIS COMMUNICATION.

- Extensions of time may be available under the provisions of 37 CFR 1.136(a). In no event, however, may a reply be timely filed after SIX (6) MONTHS from the mailing date of this communication.
- If NO period for reply is specified above, the maximum statutory period will apply and will expire SIX (6) MONTHS from the mailing date of this communication.
- Failure to reply within the set or extended period for reply will, by statute, cause the application to become ABANDONED (35 U.S.C. § 133). Any reply received by the Office later than three months after the mailing date of this communication, even if timely filed, may reduce any earned patent term adjustment. See 37 CFR 1.704(b).

Status

1)☒ Responsive to communication(s) filed on *30 November 2007*.

2a)☒ This action is **FINAL**. 2b)☐ This action is non-final.

3)☐ Since this application is in condition for allowance except for formal matters, prosecution as to the merits is closed in accordance with the practice under *Ex parte Quayle*, 1935 C.D. 11, 453 O.G. 213.

Disposition of Claims

4)☒ Claim(s) *1-23* is/are pending in the application.

4a) Of the above claim(s) _____ is/are withdrawn from consideration.

5)☐ Claim(s) _____ is/are allowed.

6)☒ Claim(s) *1-23* is/are rejected.

7)☐ Claim(s) _____ is/are objected to.

8)☐ Claim(s) _____ are subject to restriction and/or election requirement.

Application Papers

9)☐ The specification is objected to by the Examiner.

10)☒ The drawing(s) filed on *30 November 2007, 10/30/2003* is/are: a)☒ accepted or b)☐ objected to by the Examiner.

Applicant may not request that any objection to the drawing(s) be held in abeyance. See 37 CFR 1.85(a).

Replacement drawing sheet(s) including the correction is required if the drawing(s) is objected to. See 37 CFR 1.121(d).

11)☐ The oath or declaration is objected to by the Examiner. Note the attached Office Action or form PTO-152.

Priority under 35 U.S.C. § 119

12)☐ Acknowledgment is made of a claim for foreign priority under 35 U.S.C. § 119(a)-(d) or (f).

a)☐ All b)☐ Some * c)☐ None of:

1.☐ Certified copies of the priority documents have been received.

2.☐ Certified copies of the priority documents have been received in Application No. _____.

3.☐ Copies of the certified copies of the priority documents have been received in this National Stage application from the International Bureau (PCT Rule 17.2(a)).

* See the attached detailed Office action for a list of the certified copies not received.

Attachment(s)

1) ☒ Notice of References Cited (PTO-892)
2) ☐ Notice of Draftsperson's Patent Drawing Review (PTO-948)
3) ☐ Information Disclosure Statement(s) (PTO/SB/08) Paper No(s)/Mail Date _____.
4) ☐ Interview Summary (PTO-413) Paper No(s)/Mail Date. _____ .
5) ☐ Notice of Informal Patent Application
6) ☐ Other: _____.

Detailed Action

1. The text of those sections of Title 35, U.S. Code not included in this action can be found in a prior Office action.

Response to Amendment

Claims 1 and 15 have been amended.

Claims 24-25 have been cancelled.

Claims 1-23 are now pending.

Applicant's amendment filed November 30, 2007 necessitated the new ground(s) of rejection presented in this Office action. Therefore, applicant's arguments with respect to claims 1-23 have been considered but are moot in view of the new ground(s) of rejection.

Accordingly, THIS ACTION IS MADE FINAL. See MPEP 706.07(a). Applicant is reminded of the extension of time policy as set forth in 37 CFR 1.136(a).

2. Claim 1 is rejected under 35 U.S.C. 103(a) as being unpatentable over Ezekiel ("Ezekiel"; US #5625783) in view of "Using adapters to reduce interaction complexity in reusable component-based software development"("Rine", 1999) and further in view of Michaelides (Michaelides, US2004/0181753).

As per independent claim 1, Ezekiel teaches a computer-implemented method of generating a componentized user interface, the method comprising:

(a) providing a first set of interface elements with a framework (col 9, paragraph 4 where common commands included as placeholders is interpreted as a set of interface elements);

(b) providing a second set of interface elements (abstract, add-on software components provides additional menu items is interpreted as one or more sets of interface elements) with a first plug-in that is linked to the framework (in fig 2, item 271 where add-on dll is interpreted as a plug-in);

(c) providing a third set of interface elements with a second plug-in that is linked to the framework (abstract and column 6 paragraph 2 add-on software components provides additional menu items is interpreted as one or more sets of interface elements) with a second plug-in that is linked to the framework (in fig 2, item 272 where add-on dll is interpreted as a plug-in).

(d) hosting the first plug-in and the second plug-in with a shell linked to the framework (in fig 2, items 271, 272 hosted by shell 260 linked to 250 interpreted as a framework).

(e) Although Ezekiel shows providing an interface between the shell and the first plug-in and between the shell and the second plug-in (fig 2 link between 260 and 271, 272). Ezekiel does not expressly disclose the link is a shell adapter interface, in order to utilize the second set of interface elements and the third set of interface elements, wherein the shell adapter interface maps functions of the first plug-in and functions of the second plug-in to functions of the shell.

Ezekiel does not expressly disclose the link is a shell adapter interface, in order to utilize the second set of interface elements and the third set of interface elements. However, Rine does teach this (abstract, where the components are interpreted as one or more plug-ins adapters are interpreted as one or more shell adapter interfaces).

Therefore, at the time of the invention, it would have been obvious to a person of ordinary skill in the art to use the Rine adapter in Ezekiel's method. The motivation would have been to increase the components' reusability and ease their integration (Rine, abstract).

Ezekiel as modified does not teach wherein the shell adapter interface maps functions of the first plug-in and functions of the second plug-in to functions of the shell. However, Michaelides does teach this (Michaelides[0050] the generic software adapter integrates diverse software modules by[0008] transforming source data in a source data format to target data in a target data format. This is interpreted to mean wherein the shell adapter interface maps functions of the first plug-in and functions of the second plug-in to functions of the shell).

Therefore, at the time of the invention, it would have been obvious to a person of ordinary skill in the art to use the Michaelides adapter in the Ezekiel as modified method. The motivation would have been to be cost effective (Michaelides [0004]).

3. Claims 2-5, 8-13 are rejected under 35 U.S.C. 103(a) as being unpatentable over Ezekiel, Rine, and Michaelides as applied to claim 1 above, and further in view of "Defining menus and toolbars in XML" ("Gehrman", Aug 2, 2002).

As per claim 2, Ezekiel and Rine teach the computer-implemented method of claim 1, wherein the first plug-in comprises:

(i) a first file that provides an interface between the framework and the first plug-in; and (ii) a second file (Ezekiel, col 6 paragraph 2 where the add-on is interpreted as the plug-in, the module commands object is interpreted as the interface file and the module object is interpreted as the second file).

Ezekiel does not teach that this file is written in a markup language that includes menu elements. However, Gehrman does teach this (Introduction, paragraph 2 where the XML file is interpreted as the file written in a markup language and items appearing in the menu bar may come from many different plug-ins or parts is interpreted as the XML file includes menu elements.)

Therefore, at the time of the invention, it would have been obvious to a person of ordinary skill in the art to use the Gehrman's XML in Ezekiel's method. The motivation is to facilitate customization of menus without changing the application's source code (Gehrman, introduction).

As per claim 3, although Ezekiel does teach the plug-in contains menu items (Ezekiel, abstract) Gehrman further teaches these items are included in the XML file. (Gehrman Introduction, paragraph 2 where appearance in menu bars and tool bars coded in XML is interpreted to mean the elements in the XML file includes menu bars and toolbars).

Therefore, at the time of the invention, it would have been obvious to a person of ordinary skill in the art to use the Gehrman's inclusion of menu bars and tool bars with Ezekiel's

method. The motivation is to facilitate customization of menus including menu bars and tool bars without changing the application's source code (Gehrman, introduction).

As per claim 4, Ezekiel and Rine teach the computer-implemented method of claim 1, wherein the second plug-in comprises:

(i) a first file that provides an interface between the framework and the second plug-in; and (ii) a second file (Ezekiel, col 6 paragraph 2 where the add-on is interpreted as the plug-in, the module commands object is interpreted as the interface file and the module object is interpreted as the second file).

Ezekiel does not teach that this file is written in a markup language that includes menu elements. However, Gehrman does teach this (paragraph 2 where the XML file is interpreted as the file written in a markup language and items appearing in the menu bar may come from many different plug-ins or parts is interpreted as the XML file includes menu elements.)

Therefore, at the time of the invention, it would have been obvious to a person of ordinary skill in the art to use the Gehrman's XML file in combination with Ezekiel's method. The motivation is to facilitate customization of menus without changing the application's source code (Gehrman, introduction).

As per claim 5, although Ezekiel does teach the plug-in contains menu items (Ezekiel, abstract). Gehrman further teaches these items are included in the XML file. (Gehrman, Introduction, paragraph 2 where appearance in menu bars and tool bars coded in XML is interpreted to mean the elements in the XML file include menu bars and tool bars).

Therefore, at the time of the invention, it would have been obvious to a person of ordinary skill in the art to use the Gehrman's inclusion of menu bars and toolbars with Ezekiel's method. The motivation is to facilitate customization of menus including menu bars and toolbars without changing the application's source code (Gehrman, introduction).

As per claim 8, Ezekiel and Rine teach the computer-implemented method of claim 2, wherein the first file comprises an executable file (Ezekiel, col 6 paragraph 2 where the add-on is interpreted as the plug-in, the module commands object is interpreted as the interface file and the module object is interpreted as the second file). Ezekiel does not teach the first file is an executable and the second file comprises an extensible markup language (XML).

However, Gehrman does teach this (Introduction, paragraph 2 where the actions coded in C++ is interpreted as an executable and the XML file is interpreted as the file written in an extensible markup language (XML)).

Therefore, at the time of the invention, it would have been obvious to a person of ordinary skill in the art to use the Gehrman's separation of XML file with an executable in combination with Ezekiel's method. The motivation is to facilitate customization of menus without changing the application's source code (Gehrman, introduction).

As per claim 9, Ezekiel and Rine teach the computer-implemented method of claim 2, wherein the first file comprises an executable file (Ezekiel, col 6 paragraph 2 where the add-on is interpreted as the plug-in, the module commands object is interpreted as the interface file and the module object is interpreted as the second file). Ezekiel does not teach the first file is an executable and the second file comprises a standard generalized markup language (SGML).

However, Gehrman does teach this (Introduction, paragraph 2 where the actions coded in C++ is interpreted as an executable and the XML file is interpreted as the file written in a standard generalized markup language (SGML)).

Therefore, at the time of the invention, it would have been obvious to a person of ordinary skill in the art to use the Gehrman's separation of XML file with an executable in combination with Ezekiel's method. The motivation is to facilitate customization of menus without changing the application's source code (Gehrman, introduction).

As per claim 10, Ezekiel and Rine teach the computer-implemented method of claim 4, wherein the first file comprises an executable file (Ezekiel, col 6 paragraph 2 where the add-on is interpreted as the plug-in, the module commands object is interpreted as the interface file and the module object is interpreted as the second file). Ezekiel does not teach the first file is an executable and the second file comprises an extensible markup language (XML). However, Gehrman does teach this (Introduction, paragraph 2 where the actions coded in C++ is interpreted as an executable and the XML file is interpreted as the file written in an extensible markup language (XML)).

Therefore, at the time of the invention, it would have been obvious to a person of ordinary skill in the art to use the Gehrman's separation of XML file with an executable in combination with Ezekiel's method. The motivation is to facilitate customization of menus without changing the application's source code (Gehrman, introduction).

As per claim 11, Ezekiel and Rine teach the computer-implemented method of claim 4, wherein the first file comprises an executable file (Ezekiel, col 6 paragraph 2 where the add-on is interpreted as the plug-in, the module commands object is interpreted as the interface file and the module object is interpreted as the second file). Ezekiel does not teach the first file is an executable and the second file comprises a standard generalized markup language (SGML). However, Gehrman does teach this (Introduction, paragraph 2 where the actions coded in C++ is interpreted as an executable and the XML file is interpreted as the file written in a standard generalized markup language (SGML)).

Therefore, at the time of the invention, it would have been obvious to a person of ordinary skill in the art to use the Gehrman's separation of XML file with an executable in combination with Ezekiel's method. The motivation is to facilitate customization of menus without changing the application's source code (Gehrman, introduction).

As per claim 12, Ezekiel and Rine teach the computer implemented method of claim 1, wherein the framework is configured to provide the first set of interface elements (col 9, paragraph 4 where *common commands included as placeholders* is interpreted as a set of interface elements); Ezekiel does not teach the set is for a plurality of applications. However Gehrman does teach this (Gehrman page 3, paragraph 2 where standard actions are stored in a non-application specific class.)

Therefore, at the time of the invention, it would have been obvious to a person of ordinary skill in the art to use Gehrman's separation of a set of Standard actions, which are non-

application specific. The motivation is to have the standard actions automatically be included in the application(s) without having to rewrite code. (Gehrman, page 3 paragraph 2).

As per claim 13, Ezekiel teaches the computer implemented method of claim 1, wherein the second set and the third set of interface elements comprise interface elements for the same application (Ezekiel, fig 2 items 260, 271, 272).

4. Claims 6, 7, and 14 are rejected under 35 U.S.C. 103(a) as being unpatentable over Ezekiel, Rine and Michaelides as applied to claim 1 above, and further in view of "Microsoft Office 2000/ Visual Basic: Programmer's Guide" ("Shank", April, 1999).

As per claim 6, Ezekiel and Rine teach the computer-implemented method of claim 1. Ezekiel and Rine do not teach wherein the framework is configured to discover the first plug-in and the second plug-in. However Shank does teach this (Shank chapter 2, "Deploying Templates and Application-Specific Add-ins" paragraph 3 where the default folder c:\windows\application Data\Microsoft\Addins is interpreted as the default folder the framework is setup to discover plug-ins).

Therefore, at the time of the invention, it would have been obvious to a person of ordinary skill in the art to use the Shank's default folder in Ezekiel's modified method. The motivation would have been to make it easier to locate an add-in (Shank, chapter 2, "Deploying Templates and Application-Specific Add-ins" paragraph 7).

As per claim 7, Even though Ezekiel shows the application locating and loading the plug-ins(fig 2, 260, 271, 272). He does not explicitly teach this is an automatic loading process. However Shank does teach this (Shank chapter 2, "Deploying Templates and Application-Specific Add-ins" paragraph 3 where add-ins that are installed in the add-ins folder automatically appear in the list of available add-ins dialog box is interpreted to mean the add-ins are automatically loaded from the default configured folder). Since no definition is provided for user interface component loader, it is interpreted in the broadest sense – an automatic loading of a plug-in.

Therefore, at the time of the invention, it would have been obvious to a person of ordinary skill in the art to use the Shank's component loader in Ezekiel's modified method. The motivation would have been that the users do not have to browse to find correct files. (Shank, chapter 2, "Deploying Templates and Application-Specific Add-ins" paragraph 5).

As per claim 14, Ezekiel and Rine teach the computer-implemented method of claim 1 wherein the second set of interface elements comprises interface elements for a first application (Ezekiel abstract and fig 2 items 260, 271). Ezekiel and Rine do not explicitly teach the third set of interface elements comprise interface elements for a second application that is different from the first application. However Shank does teach this (Chapter 11, Add-ins, Templates, Wizards and Libraries paragraph 3 where COM add-ins work in more than one of the applications).

Therefore, at the time of the invention, it would have been obvious to a person of ordinary skill in the art to use the Shank's non-application specific add-in in Ezekiel's method.

The motivation would have been to eliminate redundancy and be able to share code in the add-in (Shank, Chapter 11, Add-ins, Templates, Wizards and Libraries paragraph 3).

5. Claim 15 is rejected under 35 U.S.C. 103(a) as being unpatentable over Ezekiel ("Ezekiel"; US #5625783) in view of "Using adapters to reduce interaction complexity in reusable component-based software development"("Rine", 1999) and "Microsoft Office 2000/ Visual Basic: Programmer's Guide" ("Shank", April, 1999) and further in view of Michaelides ("Michaelides", US2004/0181753).

As per independent claim 15, Ezekiel teaches a computer implemented method of providing extensibility to a user interface, the method comprising:

(a) providing a framework the framework comprising a first set of interface elements (Ezekiel, col 9, paragraph 4 where *common commands included as placeholders* is interpreted as a set of interface elements). Although Ezekiel shows the locating and loading of the plug-ins(fig 2, 260, 271, 272), Ezekiel does not explicitly teach the addition of a user interface component loader, the framework configured to discover a plug-in located in a plug-in directory. However Shank does teach this (Shank chapter 2, "Deploying Templates and Application-Specific Add-ins" paragraph 3 where the default folder c:\windows\application Data\Microsoft\Addins is interpreted as the default folder the framework is setup to discover plug-ins) and (Shank chapter 2, "Deploying Templates and Application-Specific Add-ins" paragraph 3 where the add-ins that are installed in the add-ins folder automatically appear in the list of available add-ins dialog box is interpreted to mean the add-ins are automatically loaded from the default configured folder).

Since no definition is provided for user interface component loader it is interpreted in the broadest sense – an automatic loading of an add-in.

Therefore, at the time of the invention, it would have been obvious to a person of ordinary skill in the art to use the Shank's default folder and component loader in Ezekiel's method. The motivation would have been that it is easier to locate an add-in (Shank, chapter 2, "Deploying Templates and Application-Specific Add-ins" paragraph 7) and users do not have to browse to find correct files(Shank, chapter 2, "Deploying Templates and Application-Specific Add-ins" paragraph 5).

(b)Ezekiel teaches loading the plug-in, the plug-in to provide a second set of interface elements. (Ezekiel abstract and col 6 par 2, *add-on software components provide additional menu items* are interpreted as one or more sets of interface elements). Ezekiel does not teach the plug-in was loaded with a user interface component loader. Shank does (Shank chapter 2, "Deploying Templates and Application-Specific Add-ins" paragraph 3 where add-ins that are installed in the add-ins folder automatically appear in the list of available add-ins dialog box is interpreted to mean the add-ins are automatically loaded from the default configured folder). Since no definition is provided for user interface component loader it is interpreted in the broadest sense –an automatic loading of an add-in.

Therefore, at the time of the invention, it would have been obvious to a person of ordinary skill in the art to use the Shank's default folder and component loader in Ezekiel's method. The motivation would have been that users do not have to browse to find correct files(Shank, chapter 2, "Deploying Templates and Application-Specific Add-ins" paragraph 5).

(c)Ezekiel teaches hosting the plug-in with a shell linked to the framework (in fig 2, item 271 hosted by shell 260 linked to 250 interpreted as a framework).

(d) Ezekiel teaches providing an interface between the shell and the plug-in (fig 2 link between 260 and 271). Ezekiel and Shank do not expressly disclose the link is a shell adapter interface, in order to utilize the second set of interface elements. However, Rine does teach this (abstract, where the components are interpreted as one or more plug-ins and adapter is interpreted as the shell adapter interface).

Therefore, at the time of the invention, it would have been obvious to a person of ordinary skill in the art to use the Rine's adapter in Ezekiel's modified method. The motivation would have been to increase the components' reusability and ease their integration (Rine, abstract).

Ezekiel as modified does not explicitly teach wherein the shell adapter interface maps functions of the plug-in to functions of the shell. However, Michaelides does teach this (Michaelides[0050] the generic software adapter integrates diverse software modules by[0008] transforming source data in a source data format to target data in a target data format. This is interpreted to mean wherein the wherein the shell adapter interface maps functions of the plug-in to functions of the shell).

Therefore, at the time of the invention, it would have been obvious to a person of ordinary skill in the art to use the Michaelides adapter in the combination of Ezekiel's as modified method. The motivation would have been to be cost effective (Michaelides [0004]).

6. Claims 16-23 are rejected under 35 U.S.C. 103(a) as being unpatentable over Ezekiel, Rine, Shank and Michaelides as applied to claim 15 above, and further in view of "Defining menus and toolbars in XML" ("Gehrman", Aug 2, 2002).

As per claim 16, Ezekiel, Rine and Shank teach the computer-implemented method of claim 15, wherein the plug-in comprises:

(i) a first file that provides an interface between the framework and the plug-in; (ii) a second file written in a markup language and that includes menu elements (Ezekiel, col 6 paragraph 2 where the add-on is interpreted as the plug-in, the module commands object is interpreted as the interface file and the module object is interpreted as the second file). Ezekiel does not teach that this file is written in a markup language that includes menu elements. However, Gehrman does teach this (Introduction, paragraph 2 where the XML file is interpreted as the file written in a markup language and items appearing in the menu bar may come from many different plug-ins or parts is interpreted as the XML file includes menu elements.)

Therefore, at the time of the invention, it would have been obvious to a person of ordinary skill in the art to use the Gehrman's XML in Ezekiel's method. The motivation is to facilitate customization of menus without changing the application's source code (Gehrman, introduction).

As per claim 17, Ezekiel, Rine, Shank in view of Gehrman teach the computer-implemented method of claim 16, wherein the menu elements are selected from the group consisting of a toolbar, a status bar, and a menu bar. Although Ezekiel does teach the plug-in

contains menu items (Ezekiel, abstract) Gehrman further teaches these items are included in the XML file. (Gehrman Introduction, paragraph 2 where appearance in menu bars and tool bars coded in XML is interpreted to mean the elements in the XML file include menu bars and toolbars).

Therefore, at the time of the invention, it would have been obvious to a person of ordinary skill in the art to use the Gehrman's inclusion of menu bars and toolbars with Ezekiel's method. The motivation is to facilitate customization of menus including menu bars and toolbars without changing the application's source code (Gehrman, introduction).

As per claim 18, Ezekiel, Rine, Shank and Gehrman teach the computer-implemented method of claim 16, wherein the first file comprises an executable file (Ezekiel, col 6 paragraph 2 where the add-on is interpreted as the plug-in, the module commands object is interpreted as the interface file and the module object is interpreted as the second file). Ezekiel does not teach the first file is an executable and the second file comprises an extensible markup language (XML). However, Gehrman does teach this (Introduction, paragraph 2 where the actions coded in C++ is interpreted as an executable and the XML file is interpreted as the file written in an extensible markup language (XML)).

Therefore, at the time of the invention, it would have been obvious to a person of ordinary skill in the art to use the Gehrman's separation of XML file with an executable in combination with Ezekiel's method. The motivation is to facilitate customization of menus without changing the application's source code (Gehrman, introduction).

As per claim 19, Ezekiel, Rine, Shank and Gehrman teach the computer-implemented method of claim 16, wherein the first file comprises an executable file (Ezekiel, col 6 paragraph 2 where the add-on is interpreted as the plug-in, the module commands object is interpreted as the interface file and the module object is interpreted as the second file). Ezekiel does not teach the first file is an executable and the second file comprises a standard generalized markup language (SGML). However, Gehrman does teach this (Introduction, paragraph 2 where the actions coded in C++ is interpreted as an executable and the XML file is interpreted as the file written in a standard generalized markup language (SGML)).

Therefore, at the time of the invention, it would have been obvious to a person of ordinary skill in the art to use the Gehrman's separation of XML file with an executable in combination with Ezekiel's method. The motivation is to facilitate customization of menus without changing the application's source code (Gehrman, introduction).

As per claim 20, Ezekiel, Rine, Shank and Gehrman teach the computer implemented method of claim 15, wherein the framework is configured to provide the first set of interface elements (col 9, paragraph 4 where *common commands included as placeholders* is interpreted as a set of interface elements); Ezekiel does not teach the set is for a plurality of applications. However Gehrman does teach this (Gehrman page 3, paragraph 2 where standard actions are stored in a non-application specific class.)

Therefore, at the time of the invention, it would have been obvious to a person of ordinary skill in the art to use Gehrman's separation of a set of standard actions, which are non-

application specific. The motivation is to have the standard actions automatically be included in the application(s) without having to rewrite code. (Gehrman, page 3 paragraph 2).

As per claim 21, Ezekiel, Rine, and Shank teach the computer-implemented method of claim 15, wherein the method further comprises:

(e) Ezekiel shows loading a second plug-in (Ezekiel fig 2, 260, 271, 272) providing a third set of interface elements (Ezekiel, abstract). Ezekiel does not explicitly teach the addition of a user interface component loader. However Shank does teach this (Shank chapter 2, "Deploying Templates and Application-Specific Add-ins" paragraph 3 where add-ins interpreted as plug-ins that are installed in the add-ins folder automatically appear in the list of available add-ins dialog box is interpreted to mean the add-ins are automatically loaded from the default configured folder). Since no definition is provided for user interface component loader it is interpreted in the broadest sense –an automatic loading of an add-in.

Therefore, at the time of the invention, it would have been obvious to a person of ordinary skill in the art to use the Shank's default folder and component loader in Ezekiel's method. The motivation would have been that users do not have to browse to find correct files(Shank, chapter 2, "Deploying Templates and Application-Specific Add-ins" paragraph 5).

(f) Ezekiel teaches hosting the second plug-in with a shell linked to the framework (in fig 2, items 271, 272 hosted by shell 260 linked to 250 interpreted as a framework)

(g) Ezekiel teaches providing an interface between the shell and the second plug-in with a second shell adapter interface in order to utilize the third set of interface elements. (fig 2 link between 260 and 271, 272). Ezekiel does not expressly disclose the link is a shell adapter

interface, in order to utilize the third set of interface elements. However, Rine does teach this (abstract, where the components are interpreted as one or more plug-ins and adapters are interpreted as one or more shell adapter interfaces).

Therefore, at the time of the invention, it would have been obvious to a person of ordinary skill in the art to use the Rine adapter in Ezekiel's method. The motivation would have been to increase the components' reusability and ease their integration (Rine, abstract).

As per claim 22, Ezekiel teaches the computer-implemented method of claim 21, wherein the second set and the third set of interface elements comprise interface elements for the same application (Ezekiel, fig 2 items 260, 271, 272).

As per claim 23, Ezekiel, Rine, and Shank teach the computer-implemented method of claim 21 wherein the second set of interface elements comprises interface elements for a first application (Ezekiel fig2 items 260, 271). Ezekiel does not explicitly teach the third set of interface elements comprise interface elements for a second application that is different from the first application. However Shank does teach this (Chapter 11, Add-ins, Templates, Wizards and Libraries paragraph 3 where COM add-ins work in more than one of the applications).

Therefore, at the time of the invention, it would have been obvious to a person of ordinary skill in the art to use the Shank's non-application specific add-in in Ezekiel's method. The motivation would have been to eliminate redundancy and be able to share code in the add-in (Shank, Chapter 11, Add-ins, Templates, Wizards and Libraries paragraph 3 where add-in is interpreted as plug-in).

7. Applicant's arguments with respect to the pending claims as amended in the 30 November 2007 response have been considered but are moot in view of the new ground(s) of rejection.

Action is Final

8. **THIS ACTION IS FINAL**. Applicant is reminded of the extension of time policy as set forth in 37 CFR 1.136(a).

A shortened statutory period for reply to this final action is set to expire THREE MONTHS from the mailing date of this action. In the event a first reply is filed within TWO MONTHS of the mailing date of this final action and the advisory action is not mailed until after the end of the THREE-MONTH shortened statutory period, then the shortened statutory period will expire on the date the advisory action is mailed, and any extension fee pursuant to 37 CFR 1.136(a) will be calculated from the mailing date of the advisory action. In no event, however, will the statutory period for reply expire later than SIX MONTHS from the mailing date of this final action.

Conclusion

9. Any inquiry concerning this communication or earlier communications from the examiner should be directed to R. M. Herbst whose telephone number is (571) 270-5132. The examiner can normally be reached on Monday - Thursday from 9:00 AM to 4:00 PM ET.

If attempts to reach the examiner by telephone are unsuccessful, the examiner's supervisor, Taghi Arani can be reached on 571-272-3787. The fax phone number for the organization where this application or proceeding is assigned is 571-273-8300.

Information regarding the status of an application may be obtained from the Patent Application Information Retrieval (PAIR) system. Status information for published applications may be obtained from either Private PAIR or Public PAIR. Status information for unpublished applications is available through Private PAIR only. For more information about the PAIR system, see http://pair-direct.uspto.gov. Should you have questions on access to the Private PAIR system, contact the Electronic Business Center (EBC) at 866-217-9197 (toll-free). If you would like assistance from a USPTO Customer Service Representative or access to the automated information system, call 800-786-9199 (IN USA OR CANADA) or 571-272-1000.

rmh

/Taghi T. Arani/
Supervisory Patent Examiner, Art Unit 4121
1/3/2007